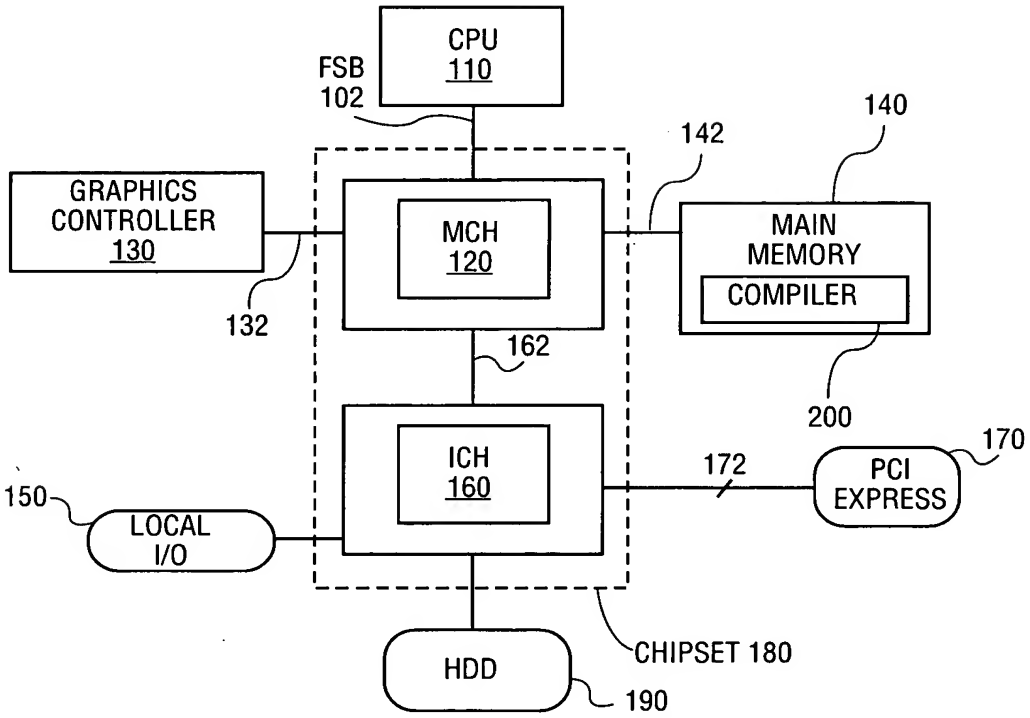
CPU
110
FSB
102
140
142
GRAPHICS
CONTROLLER
130
MCH
120
MAIN
MEMORY
COMPILER
132
162
200
ICH
160
172
PCI
EXPRESS
170
150
LOCAL
I/O
CHIPSET 180
HDD
190
COMPUTER SYSTEM
100

FIG. 1

valid Ingress_PPS ()
{
//the PPS loop
for (;;)
{
Receive_Packet ();
IPv4_Forward ()
Enqueue_packet ();
}
}
SEQUENTIAL PPS 280

FIG. 2A

valid Ingress_PPS_Stage1 ()
{
//the PPS loop
for (;;)
{
Receive_Packet ();
}
}
data 1
valid Ingress_PPS_Stage2 ()
{
//the PPS loop
for (;;)
{
IPv4_Forward ()
}
}
data 2
valid Ingress_PPS_Stage3 ()
{
//the PPS loop
for (;;)
{
Enqueue_packet ()
}
}
PIPELINED PPS 300

FIG. 2B

```
void MyPPS()
{
   for (;;)
   {
     if (p ())
     {
        x = f1 ();
        y = g1 ();
     }
      else
     {
        x = f2 ();
        y = g2 ();
     }
      z = h (x,y);
  }
}                          290
```

FIG. 3A

```
void MyPPS2()
{
  for (;;)
   {
     if (p ())
     {
        x1 = f1 ();
        y1 = g1 ();
     }
      else
     {
        x2 = f2 ();
        y2 = g2 ();
     }
      x = phi (x1, x2);
      y = phi (y1, y2)
      z = h (x, y);
  }
}                          400
```

FIG. 4

```
void MyPPS2_Stage1 ()
{
   for (;;)
   {
      if (p ())
      {
         c = 1;
         x = f1 ();
      }
       else
      {
        c = 2;
        x = f2 () ;
      }
       Send_To_Stage2 (c, x);
   }
                           310
```

x, c

```
void MyPPS2_Stage2 ()
{
   for (;;)
   {
      Receive_From_Stage1 (&c, &x);
      if (C ==1)
      {
         y = g1 ();
      }
       else
      {
        y = g2 ();
      }
        z = h (x,y);
   }
}
                           320
```

FIG. 3B

FIG. 3C

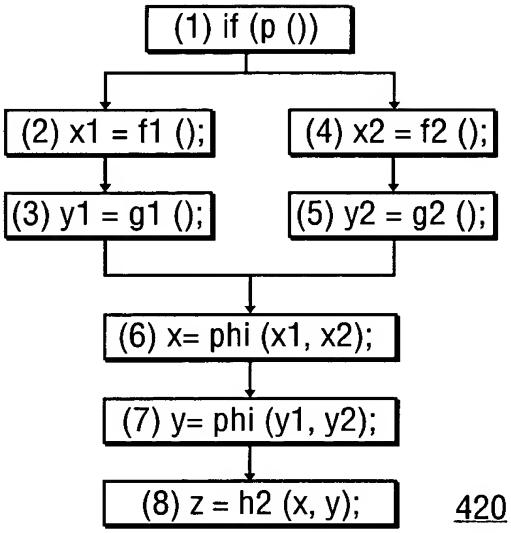
(1) if (p ())
(2) x1 = f1 ();
(4) x2 = f2 ();
(3) y1 = g1 ();
(5) y2 = g2 ();
(6) x= phi (x1, x2);
(7) y= phi (y1, y2);
(8) z = h2 (x, y);
420

FIG. 5

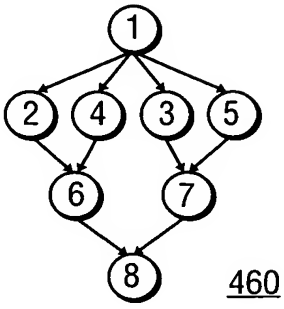
1
2
4
3
5
6
7
8
460

FIG. 6

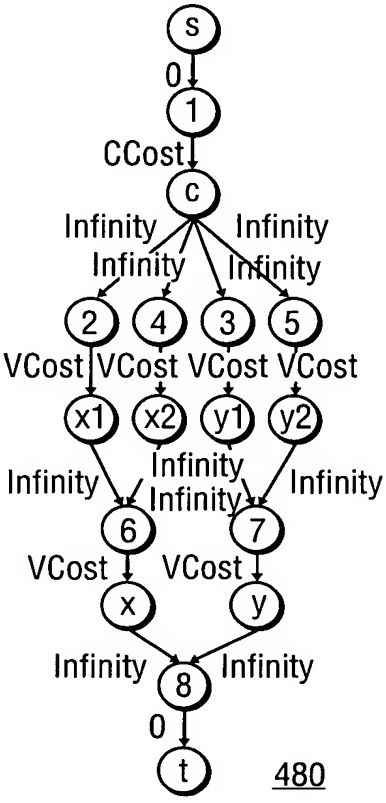
s
0
1
CCost
c
Infinity
Infinity
Infinity
Infinity
2
4
3
5
VCost
VCost
VCost
VCost
x1
x2
y1
y2
Infinity
Infinity
Infinity
Infinity
6
7
VCost
VCost
x
y
Infinity
Infinity
8
0
t
480

FIG. 7

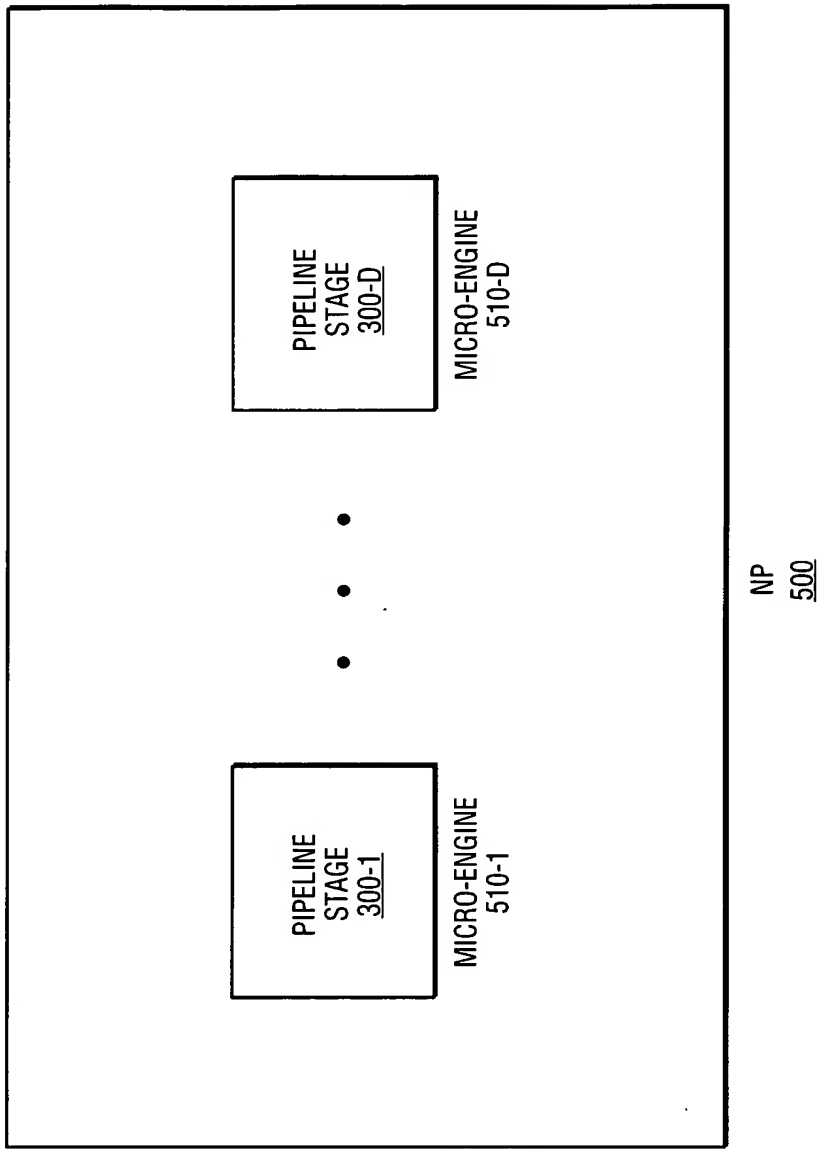
PIPELINE STAGE 300-D
MICRO-ENGINE 510-D
PIPELINE STAGE 300-1
MICRO-ENGINE 510-1
NP 500

FIG. 8

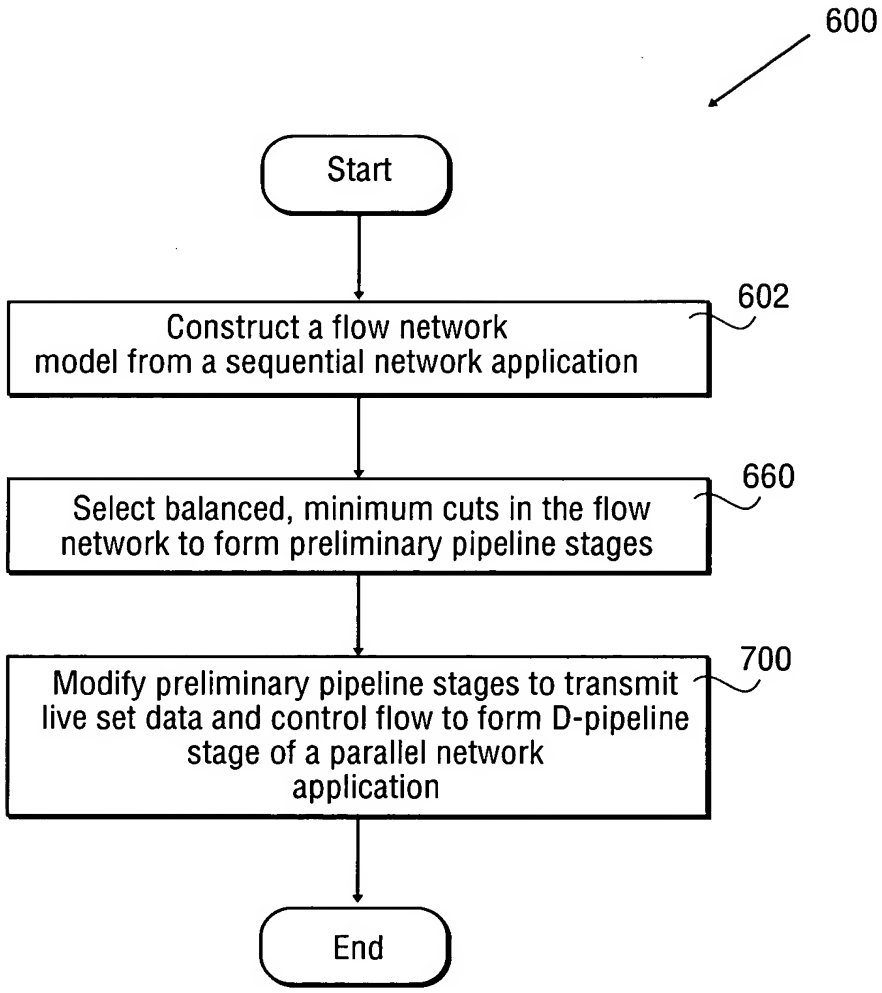
600
Start
Construct a flow network model from a sequential network application
602
Select balanced, minimum cuts in the flow network to form preliminary pipeline stages
660
Modify preliminary pipeline stages to transmit live set data and control flow to form D-pipeline stage of a parallel network application
700
End

FIG. 9

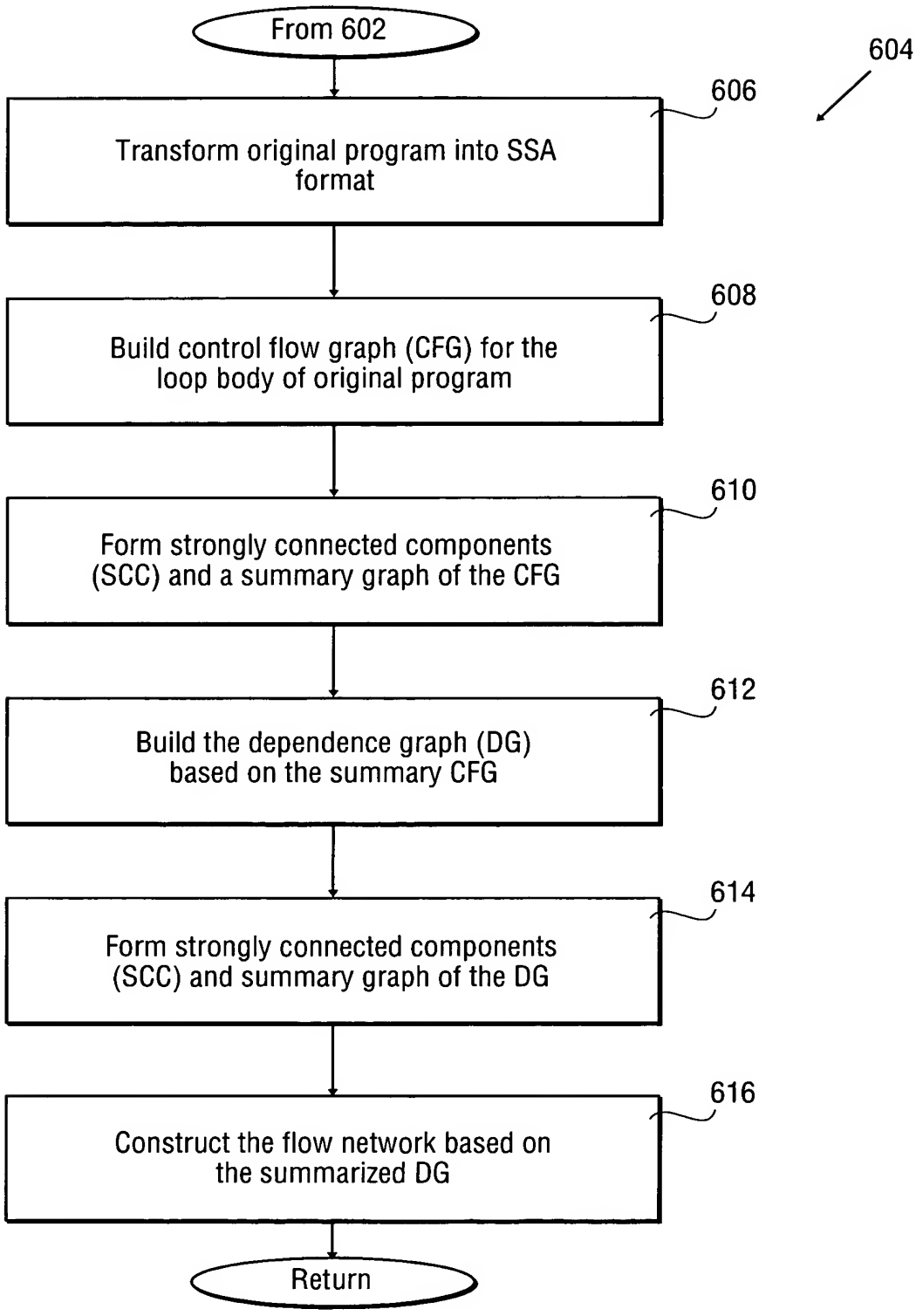
From 602
604
606
Transform original program into SSA format
608
Build control flow graph (CFG) for the loop body of original program
610
Form strongly connected components (SCC) and a summary graph of the CFG
612
Build the dependence graph (DG) based on the summary CFG
614
Form strongly connected components (SCC) and summary graph of the DG
616
Construct the flow network based on the summarized DG
Return

FIG. 10

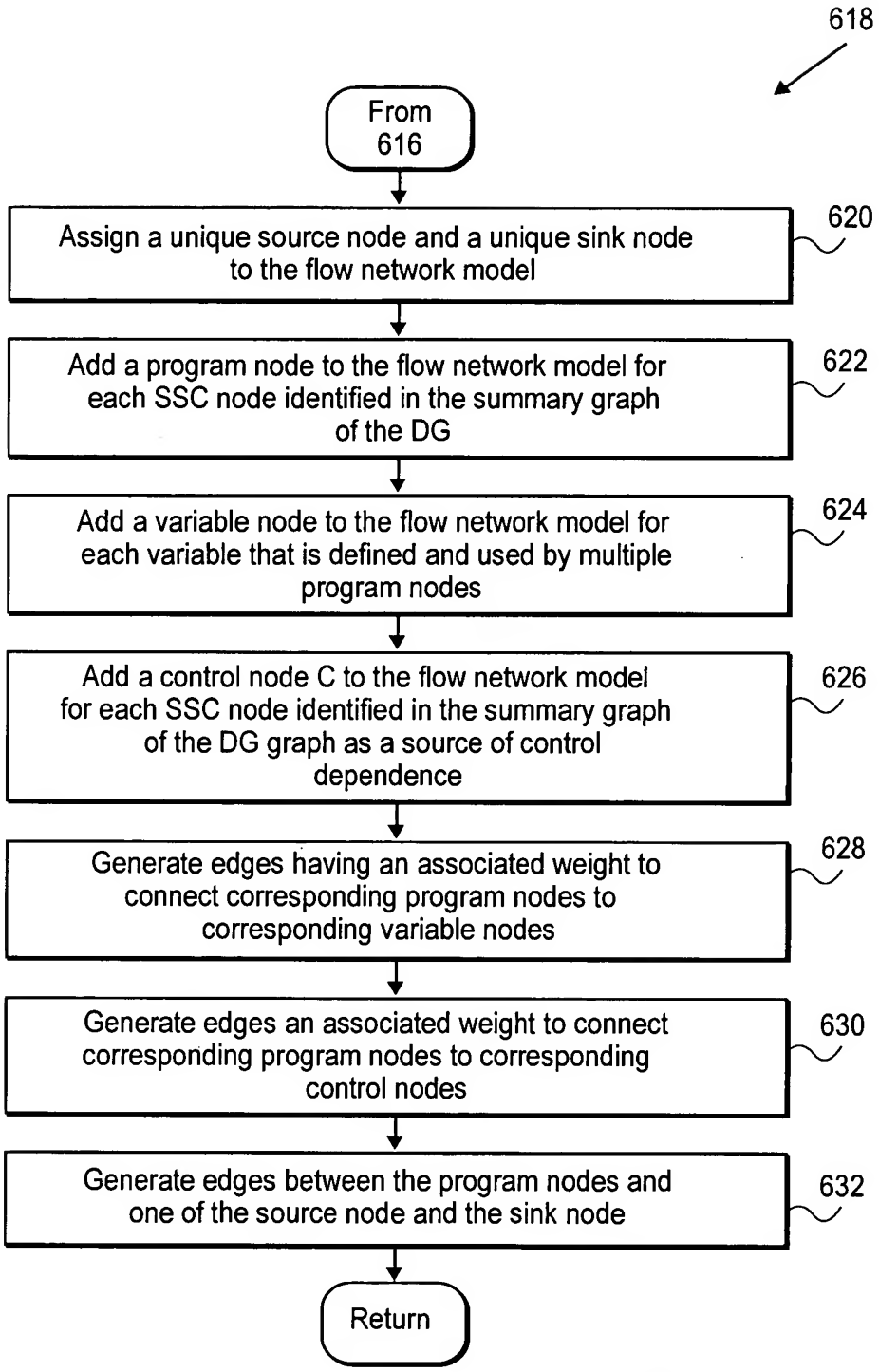
618
From 616
Assign a unique source node and a unique sink node to the flow network model
620
Add a program node to the flow network model for each SSC node identified in the summary graph of the DG
622
Add a variable node to the flow network model for each variable that is defined and used by multiple program nodes
624
Add a control node C to the flow network model for each SSC node identified in the summary graph of the DG graph as a source of control dependence
626
Generate edges having an associated weight to connect corresponding program nodes to corresponding variable nodes
628
Generate edges an associated weight to connect corresponding program nodes to corresponding control nodes
630
Generate edges between the program nodes and one of the source node and the sink node
632
Return

FIG. 11

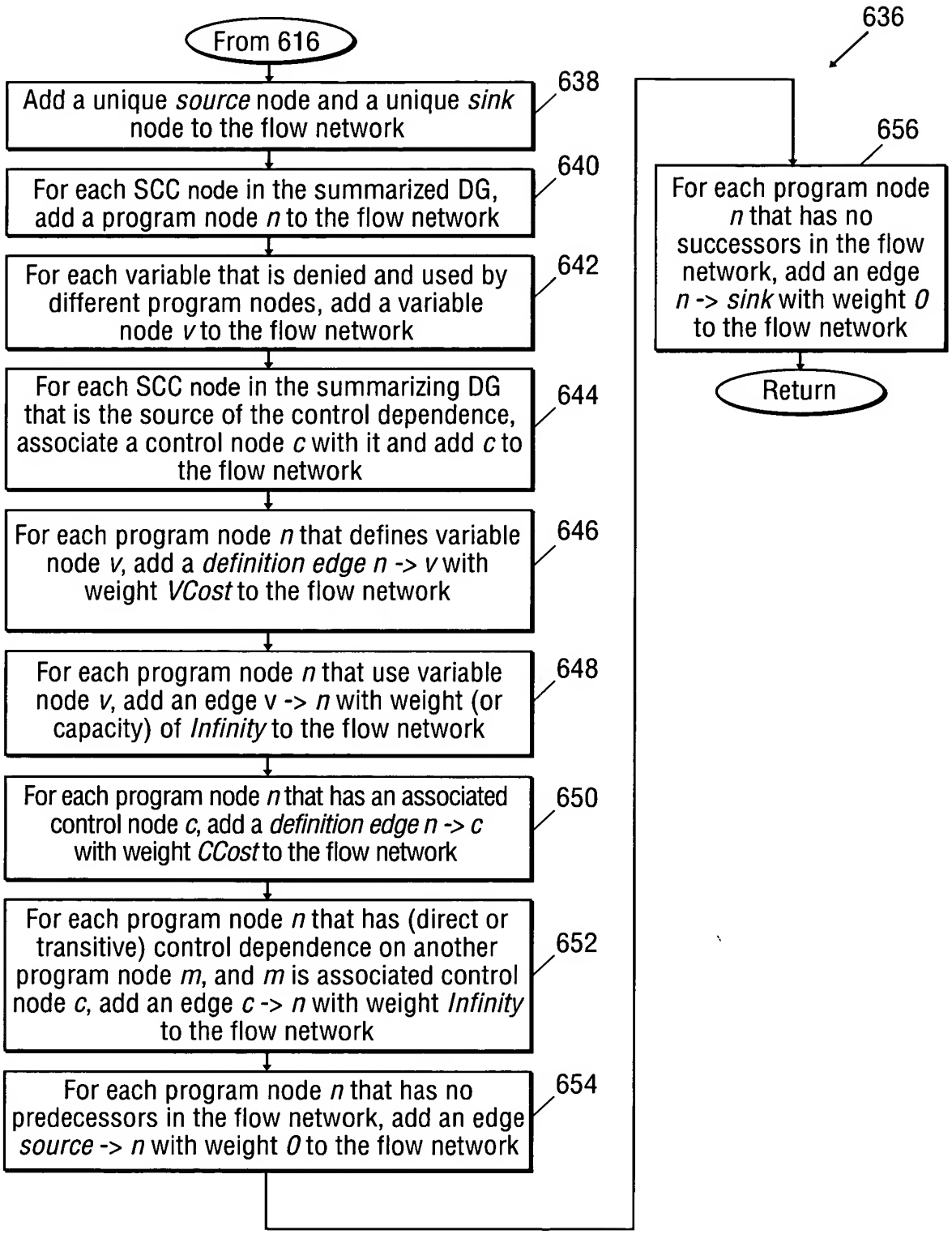
From 616
Add a unique *source* node and a unique *sink* node to the flow network
638
For each SCC node in the summarized DG, add a program node *n* to the flow network
640
For each variable that is denied and used by different program nodes, add a variable node *v* to the flow network
642
For each SCC node in the summarizing DG that is the source of the control dependence, associate a control node *c* with it and add *c* to the flow network
644
For each program node *n* that defines variable node *v*, add a *definition edge n -> v* with weight *VCost* to the flow network
646
For each program node *n* that use variable node *v*, add an edge v -> *n* with weight (or capacity) of *Infinity* to the flow network
648
For each program node *n* that has an associated control node *c*, add a *definition edge n -> c* with weight *CCost* to the flow network
650
For each program node *n* that has (direct or transitive) control dependence on another program node *m*, and *m* is associated control node *c*, add an edge *c -> n* with weight *Infinity* to the flow network
652
For each program node *n* that has no predecessors in the flow network, add an edge *source -> n* with weight *0* to the flow network
654
636
656
For each program node *n* that has no successors in the flow network, add an edge *n -> sink* with weight *0* to the flow network
Return

FIG. 12

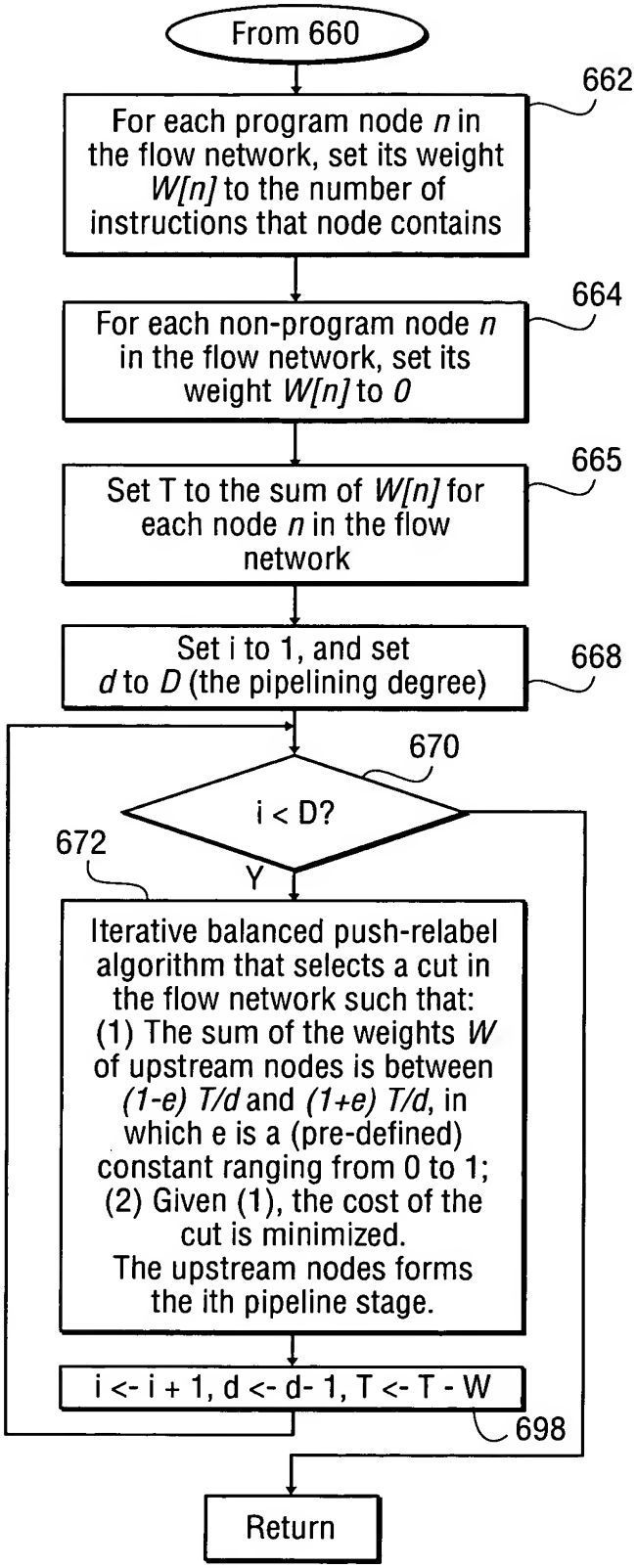
From 660
661
662
For each program node n in the flow network, set its weight W[n] to the number of instructions that node contains
664
For each non-program node n in the flow network, set its weight W[n] to 0
665
Set T to the sum of W[n] for each node n in the flow network
Set i to 1, and set d to D (the pipelining degree)
668
670
i < D?
672
Y
Iterative balanced push-relabel algorithm that selects a cut in the flow network such that:
(1) The sum of the weights W of upstream nodes is between (1-e) T/d and (1+e) T/d, in which e is a (pre-defined) constant ranging from 0 to 1;
(2) Given (1), the cost of the cut is minimized.
The upstream nodes forms the ith pipeline stage.
i <- i + 1, d <- d- 1, T <- T - W
698
Return

FIG. 13

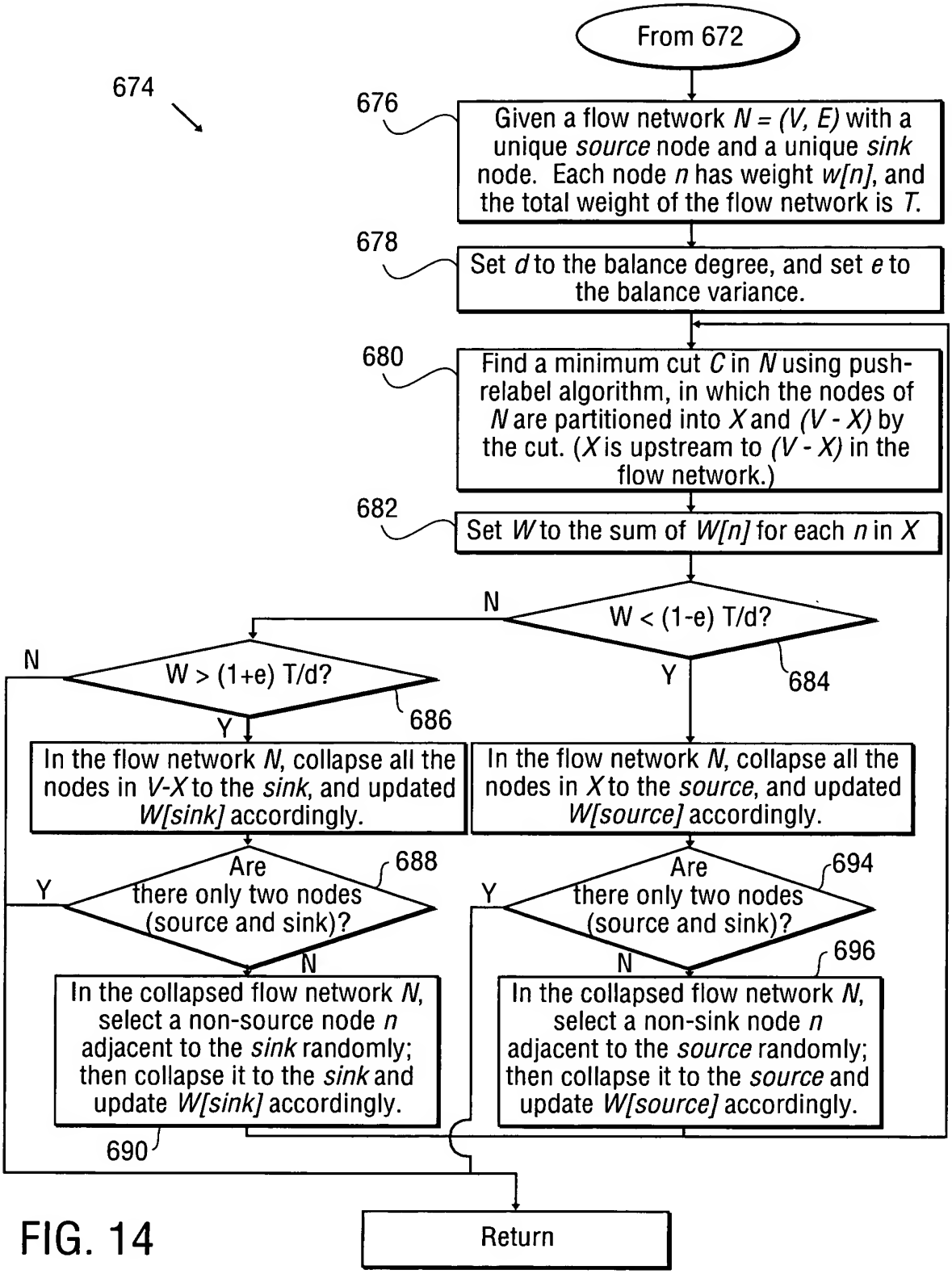
From 672
674
676
Given a flow network N = (V, E) with a unique source node and a unique sink node. Each node n has weight w[n], and the total weight of the flow network is T.
678
Set d to the balance degree, and set e to the balance variance.
680
Find a minimum cut C in N using push-relabel algorithm, in which the nodes of N are partitioned into X and (V - X) by the cut. (X is upstream to (V - X) in the flow network.)
682
Set W to the sum of W[n] for each n in X
N
W < (1-e) T/d?
684
Y
N
W > (1+e) T/d?
686
Y
In the flow network N, collapse all the nodes in V-X to the sink, and updated W[sink] accordingly.
In the flow network N, collapse all the nodes in X to the source, and updated W[source] accordingly.
Y
Are there only two nodes (source and sink)?
688
N
Y
Are there only two nodes (source and sink)?
694
N
In the collapsed flow network N, select a non-source node n adjacent to the sink randomly; then collapse it to the sink and update W[sink] accordingly.
In the collapsed flow network N, select a non-sink node n adjacent to the source randomly; then collapse it to the source and update W[source] accordingly.
696
690
Return

FIG. 14

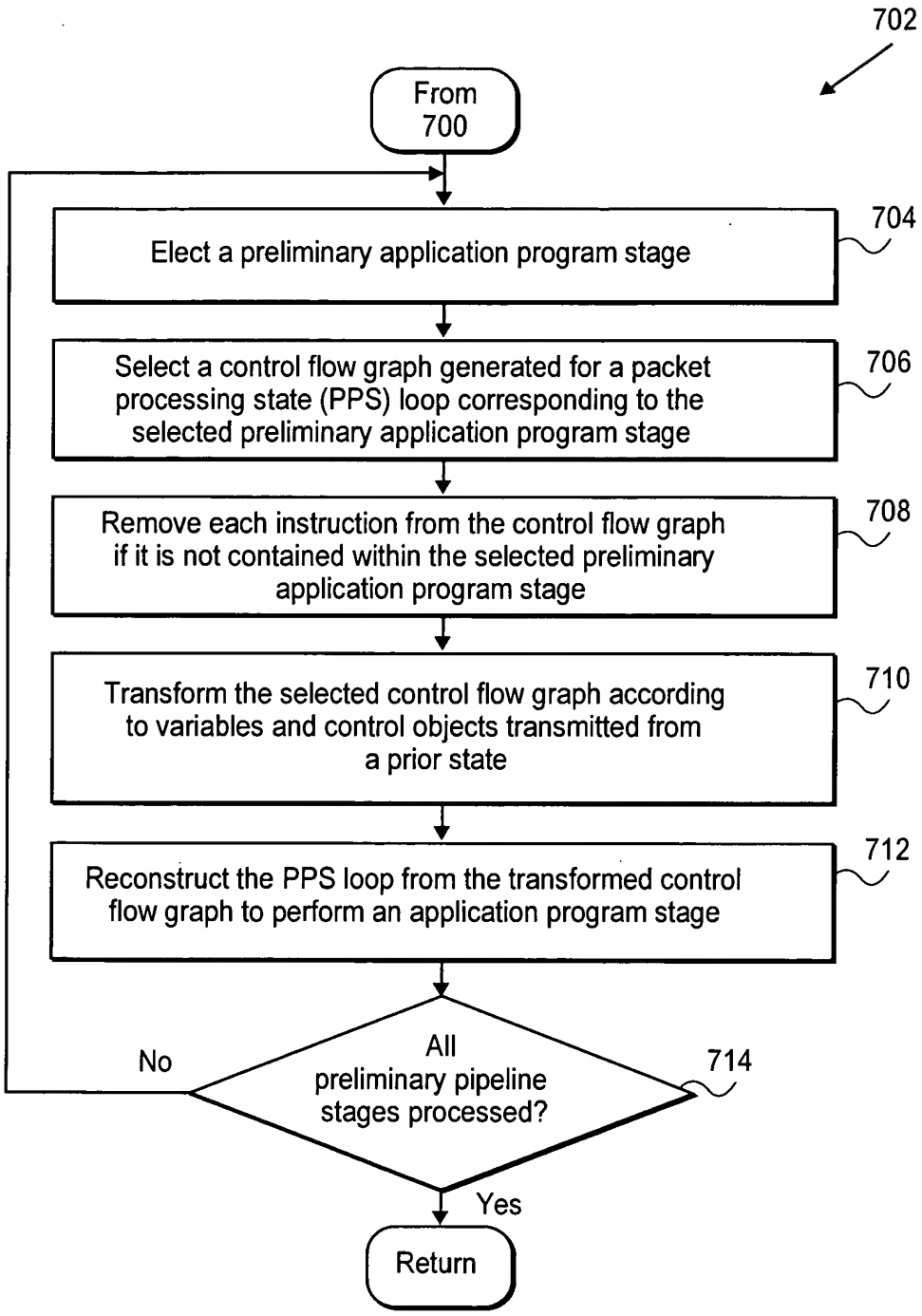
702
From 700
Elect a preliminary application program stage
704
Select a control flow graph generated for a packet processing state (PPS) loop corresponding to the selected preliminary application program stage
706
Remove each instruction from the control flow graph if it is not contained within the selected preliminary application program stage
708
Transform the selected control flow graph according to variables and control objects transmitted from a prior state
710
Reconstruct the PPS loop from the transformed control flow graph to perform an application program stage
712
All preliminary pipeline stages processed?
714
No
Yes
Return

FIG. 15

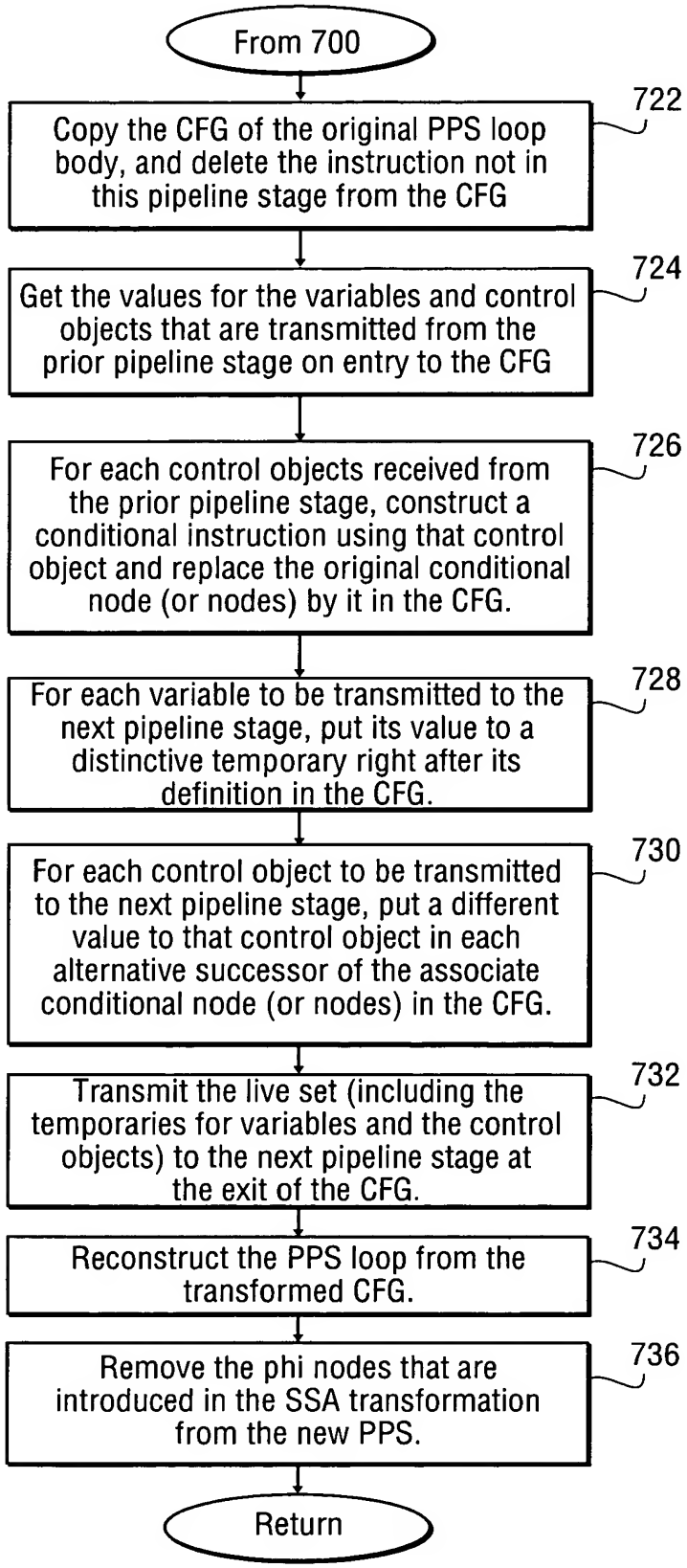
From 700
720
722
Copy the CFG of the original PPS loop body, and delete the instruction not in this pipeline stage from the CFG
724
Get the values for the variables and control objects that are transmitted from the prior pipeline stage on entry to the CFG
726
For each control objects received from the prior pipeline stage, construct a conditional instruction using that control object and replace the original conditional node (or nodes) by it in the CFG.
728
For each variable to be transmitted to the next pipeline stage, put its value to a distinctive temporary right after its definition in the CFG.
730
For each control object to be transmitted to the next pipeline stage, put a different value to that control object in each alternative successor of the associate conditional node (or nodes) in the CFG.
732
Transmit the live set (including the temporaries for variables and the control objects) to the next pipeline stage at the exit of the CFG.
734
Reconstruct the PPS loop from the transformed CFG.
736
Remove the phi nodes that are introduced in the SSA transformation from the new PPS.
Return

FIG. 16